Федеральное государственное автономное
образовательное учреждение высшего образования
«СИБИРСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»
Институт математики и фундаментальной информатики
Базовая кафедра вычислительных и информационных технологий

УТВЕРЖДАЮ
Заведующий кафедрой
_____ /В. В.Шайдуров
«16» июня 2017 г.

БАКАЛАВРСКАЯ РАБОТА

Направление 02.03.01 Математика и компьютерные науки

ДИСКРЕТНЫЙ ПОДХОД ПРИ КЛАССИФИКАЦИИ ТЕКСТОВ

Научный руководитель
кандидат философских наук,
доцент _____ / Б.В.Олейников
16.06.2017

Выпускник _____ / И.А.Понамарев
16.06.2017

Красноярск 2017

РЕФЕРАТ

Выпускная квалификационная работа по теме «Дискретный подход при классификации текстов» содержит 63 страницы текстового документа, 5 приложения, 12 использованных источников.

КЛАССИФИКАЦИЯ ТЕКСТОВЫХ ДОКУМЕНТОВ С МАТЕМАТИЧЕСКИМИ ФОРМУЛАМИ, ДИСКРЕТНАЯ МОДЕЛЬ, ОСОБЫЕ ОБЪЕКТЫ, ПЕРЕВОД ЗВУКОВОГО СИГНАЛА В ТЕКСТ, ПРЕОБРАЗОВАНИЕ ДОКУМЕНТА В TEX ФОРМАТ.

Объект работы – текстовый документ с математическими формулами.

Цели:

- применение известных методов классификации документов;

- проверка влияния формул на качество классификации;

- опробирование метода перевода звукового сигнала в текст для преобразования математических формул;

- опробирование способа перевода текстового документа в формат TeX

- сравнение качества классификации при применении каждого из методов

В результате применения методов преобразования формул, которые стоит рассматривать в качестве предобработки было замечено увеличение количества классифицируемых текстов, содержащих математические формулы. Более эффективным показал себя метод Speech-To-Text, который имел более качественные результаты классификации по сравнению с способом перевода текстового документа в TeX формат.

В итоге предложенные методы преобразования формул могут дополнить арсенал уже существующих в дискретной модели методов классификации.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

В настоящее время в связи со взрывным характером порождения цифровых текстовых документов (интернет, автоматизированный документооборот, цифровые библиотеки, образовательные сайты и порталы и т.п.) все более насущной является проблема их поиска. Основополагающую роль при построении тематического полнотекстового поиска документов играет классификация.

Классификация документов – одна из задач информационного поиска, заключающаяся в отнесении документа к одной из нескольких категорий на основании содержания документа и может применятся в решении многих практических задач, таких как:

- фильтрация документов;
- распознавание спама;
- автоматическое аннотирование;
- поиск текстовых документов;
- навигация по большим информационным ресурсам;
- подбор рекламы;
- составление интернет-каталогов;
- классификация новостей;
- библиотечный УДК и ББК;
- индексация данных в поисковых запросах и т.д.

Классификация может осуществляться полностью вручную, либо автоматически с помощью созданного вручную набора правил, либо автоматически с применением методов машинного обучения.

Способы и методы классификации включаются в направление, называемое TextMining.

Сейчас TextMining активно развивается: ведутся исследования, запускаются проекты и конкурсы на выявление лучших по точности алгоритмов.

1 Подходы к классификации текстов

Существует три подхода к задаче классификации текстов.

Первый подход – это ручная классификация. Она производится по многим критериям, известным только человеку-классификатору. Подобная *ручная классификация* дорога и неприменима в случаях, когда необходимо классифицировать большое количество документов с высокой скоростью.

Второй подход заключается в *написании правил*, по которым можно отнести текст к той или иной категории. Например, одно из таких правил может выглядеть следующим образом: «если текст содержит слова ‘производная’ и ‘уравнение’, то отнести его к категории математика». Специалист, знакомый с предметной областью и обладающий навыком написания регулярных выражений, может составить ряд правил, которые затем автоматически применяются к поступающим документам для их классификации. Этот подход лучше предыдущего, поскольку процесс классификации автоматизируется и, следовательно, количество обрабатываемых документов практически не ограничено. Более того, построение правил вручную может дать лучшую точность классификации, чем при машинном обучении (см. ниже). Однако создание и поддержание правил в актуальном состоянии (например, если для классификации новостей используется имя действующего президента страны, соответствующее правило нужно время от времени изменять) требует постоянных усилий специалиста.

Третий подход основывается на *машинном обучении*. В этом подходе набор правил или, более обще, критерий принятия решения текстового классификатора, вычисляется автоматически из обучающих данных (другими словами, производится обучение классификатора). Обучающие данные – это некоторое количество хороших образцов документов из каждого класса. В

машинном обучении сохраняется необходимость ручной разметки (термин *разметка* означает процесс приписывания класса документу). Но разметка является более простой задачей, чем написание правил. Кроме того, разметка может быть произведена в обычном режиме использования системы. Например, в программе электронной почты может существовать возможность помечать письма как спам, тем самым формируя обучающее множество для классификатора – фильтра нежелательных сообщений. Таким образом, классификация текстов, основанная на машинном обучении, является примером обучения с учителем, где в роли учителя выступает человек, задающий набор классов и размечающий обучающее множество.

2 Классификация текстовых документов (дискретная модель)

2.1 Постановка задачи

Имеется множество категорий (классов, меток) $\mathfrak{E} = \{c_1, \dots, c_{|\mathfrak{E}|}\}$, множество документов $\mathfrak{D} = \{d_1, \dots, d_{|\mathfrak{D}|}\}$ и неизвестная целевая функция $\phi: \mathfrak{E} \times \mathfrak{D} \to \{0,1\}$.

Необходимо построить классификатор ϕ', максимально близкий к ϕ. Имеется некоторая начальная коллекция размеченных документов $\mathfrak{R} \subset \mathfrak{C} \times \mathfrak{D}$, для которых известны значения ϕ. Обычно её делят на «обучающую» и «проверочную» части. Первая используется для обучения классификатора, вторая — для независимой проверки качества его работы. Классификатор может выдавать точный ответ $\phi': \mathfrak{E} \times \mathfrak{D} \to \{0,1\}$ или степень подобия $\phi: \mathfrak{E} \times \mathfrak{D} \to [0,1]$.

Основная модель представления текста — это вектор в дискретном пространстве выделенных и нормализованных, т. е. специальным образом приведенных [8] к каноническому виду слов некоторого словаря: $d_i = (w_1, w_2, \dots, w_n)$, где d_i – векторное представление i-го документа, w_i – вес j-го слова в документе, n – общее количество различных слова во всех документах коллекции. На данной модели базируются практически все известные методы классификации текстов независимо от весовых предпочтений и подходов к агрегированию.

2.2 Представление документа

Перед тем, как приступать к любой задаче классификации, одна из наиболее фундаментальных процедур, которую необходимо выполнить, это выбрать представление документа и отобрать признаки. Хоть отбор признаков и применяется довольно часто в других задачах классификации,

он особенно важен в задаче классификации текстов ввиду высокой размерности (большого количества признаков) и наличия нерелевантных (шумовых) признаков. В большинстве случаев, представление текста происходит одним из двух способов.

Первый – документ как набор слов, в котором документу сопоставляются слова и частота их встречаемости в нем. Такое представление, как видим, независимо от порядка слов, в котором они встречаются в тексте.

Второй метод состоит в представлении текста как набор строк. Большинство алгоритмов классификации текстов используют первое представление ввиду его простоты и удобства для задач классификации.

Наиболее популярными способами отбора признаков являются удаление стоп-слов и стэмминг. При удалении стоп-слов, мы определяем общие слова документов, которые не являются специфичными или разделяющими для разных классов. При стэмминге, разные формы одного слова объединяются в одно слово (терм). Например, так объединяются слова разного рода/формы/времени/падежа и пр.

Выделение слов может производиться с применением различных фильтров.

Например, могут исключаться все не значащие слова, такие как местоимения, союзы, предлоги и т.п. Для научных текстов дополнительно могут исключаться слова общей лексики. Нормировка слов (приведение их к определенному виду) зависит от задачи и может включать в себя следующие этапы: для существительных – приведение к именительному падежу, для глаголов – к форме инфинитива, для прилагательных – выделение корневых форм, для синонимов – использование словаря синонимов и т.п.

На данной модели базируются практически все известные методы классификации текстов независимо от весовых предпочтений и подходов к агрегированию.

На подготовительном этапе также могут рассматриваться преобразования признаков с помощью некоторых функций, определение наиболее информативных признаков, или наоборот, отбрасывание неинформативных признаков, задачи уменьшения признакового пространства

2.3 Общий подход к классификации текстов в дискретной модели

Задано n-мерное пространство с определенной на нем метрикой. В данном пространстве текст представим, как вектор – точка в пространстве признаков (слов). Для более точного определения положения вектора необходимо выявить более информативные признаки, а также определить точки сгущения, задающие классы по обучающим выборкам. Чтобы выполнить задачу также необходимо определиться с метриками для оценки расстояния между векторами и методами классификации для объединения векторов в классы. Все это будет рассмотрено в работе дальше.

2.4 Отбор признаков

Для отбора признаков могут применяться различные методы.

Например, тривиальный отбор признаков, где входной текст разбивается на список терминов $\{t_i\}$, который в дальнейшем очищается от слов «классической» лексики, местоимений, предлогов, синонимов, однокоренных слов и т.д. На выходе получается преобразованный список терминов $\{t_i^*\}$, который будет уникальным для определенного класса.

Следующий метод – индекс Джини. Пусть $p_1(w), \ldots, p_k(w)$ – частоты наличия метки классов для слова w. Другими словами, $p_i(w)$ – условная

вероятность, что документ принадлежит классу i, если известно, что он содержит слово w. Значит, можно утверждать, что

$$\sum_{i=1}^{k} p_i(w) = 1. \quad (2.1)$$

Тогда индекс Джини для слова w, обозначаемый $G(w)$, определяется как

$$G(w) = \sum_{i=1}^{k} p_i(w)^2. \quad (2.2)$$

Также существует метод взаимной информации. Понятие меры взаимной информации произошло из теории информации, оно предоставляет формальный путь смоделировать корреляцию между признаками и классами. Заметим, что выборочная совместная встречаемость класса i и слова w в смысле взаимной информации определена как $P_i \cdot F(w)$, где $F(w)$ – часть документов, содержащая слово w, а P_i – априорная вероятность принадлежности к классу. Истинная совместная встречаемость, очевидна, равна $F(w) \cdot p_i(w)$, где $p_i(w)$ – условная вероятность принадлежности к классу i. На практике, значение $F(w) \cdot p_i(w)$ может сильно отличаться от $P_i \cdot F(w)$ в зависимости от корреляции класса i и слова w. Взаимная информация определена в терминах отношения двух выше рассмотренных величин. А именно,

$$M_i(w) = \log\left(\frac{F(w)*p_i(w)}{F(w)*P_i}\right) = \log\left(\frac{p_i(w)}{P_i}\right). \quad (2.3)$$

Как видно, при наличии слова w вероятность принадлежности к классу i возрастает, когда $M_i(w) > 0$, и наоборот – наличие слова w отрицательно сказывается на вероятности принадлежности к классу i, когда $M_i(w) < 0$.

Другая схожая мера, часто используемая в отборе признаков для документов – это прирост информации. Мера прироста информации $I(w)$ для данного слова w определяется как

$$I(w) = -\sum_{i=1}^{k} P_i * \log(P_i) + F(w) * \sum_{i=1}^{k} p_i(w) * \log\big(p_i(w)\big) + \big(1 - F(w)\big) * \sum_{i=1}^{k}\big(1 - p_i(w)\big) * \log(1 - p_i(w)). \quad (2.4)$$

Чем больше значение прироста информации $I(w)$, тем сильнее разделяющая способность слова w.

Статистика хи-квадрат (χ^2) – это еще один способ выявить зависимость (а точнее, отсутствие независимости) между словом w и определенным классом i. Как и ранее, m будет общим количеством документов в выборке, $p_i(w)$ – условная вероятность принадлежности документа к классу i при наличии слова w, P_i – априорная вероятность принадлежности к классу i, $F(w)$ – часть документов, содержащих слово w. Статистика χ^w слова между словом w и классом i определяется как

$$\chi_i^2(w) = \frac{n * F(w)^2 * (p_i(w) - P_i)^2}{F(w) * \big(1 - F(w)\big) * P_i * (1 - P_i)}. \quad (2.5)$$

2.5 Взвешивание слов

Взвешивать слова в тексте можно различными способами, от которых зависит «качество» представления текста и которые могут существенно повлиять на результат классификации текста. Для каждого терма (выделенного нормированного слова) в документе могут определяться различные числовые показатели. Наиболее популярные из них:

- частота встречаемости терма в документе – «*tf*»;
- частота встречаемости терма в других документах – «*df*»;
- длина слова;

- показатель важности термов, с которыми используется данный терм. Например, если некоторое существительное в тексте используется с прилагательными имеющими большой вес, можно также говорить об его важности.

Из этих числовых показателей можно получить функции веса термов:

- булево значение веса $w = sign(tf)$, то есть 1 – если слово встретилось в документе, 0 – иначе;

- $w = tf$ – стандартная частота слова;

- $w = \frac{tf}{df}$ – такой коэффициент часто называют «*tf-idf*» (часто применяется «сглаженная» вариация этой формулы - $w = tf * \log(\frac{1}{df})$).

2.6 Основные индексы и метрики при оценке расстояния

Во многих алгоритмах классификации текстов используется функция сравнения векторов или функция нахождения «расстояния» между векторами, которая может базироваться на метриках и индексах, таких как:

- Обобщенный индекс $r(\bar{a}, \overline{b}) = \frac{R*(n_{ij}+a*n_{ij})}{R*(n_{ij}+a*n_{ij})+n_{ij}+n_{ij}}$;

- Манхэттенская $r(\bar{a}, \overline{b}) = \sum_{i=1}^{n} |a_i - b_i|$;

- Хэмминга $r(\bar{a}, \overline{b}) = \sum_{i=1}^{n} sign|a_i - b_i|$, где a_i и b_i – двоичные вектора;

- Косинус $r(\bar{a}, \overline{b}) = \cos(g)$, где g – угол между векторами.

2.7 Методы классификации

Для классификации текстовых документов могут применяться множество различных методов. Рассмотрим некоторые из них.

2.7.1 Наивный байесовский алгоритм – это простой вероятностный классификатор, основанный на применении Теоремы Байеса со строгими предположениями о независимости [1]. Класс принадлежности c документа d считается по формуле:

$$c = argmax_{o \in c} \log\left(\frac{D_o}{D}\right) + \sum_{i \in Q} \log\left(\frac{W_{ic}+1}{n+L_c}\right), \quad (2.6)$$

где C – множество классов;

D_o – количество документов в обучающей выборке принадлежащих классу c;

D – общее количество документов в обучающей выборке;

L_c – суммарное количество слов в документах класса c в обучающей выборке;

W_{ic} – сколько раз i-ое слово встречалось в документах класса c в обучающей выборке;

Q – множество слов классифицируемого документа.

2.7.2 Алгоритм k-ближайших соседей – это алгоритм классификации, основанный на оценивании сходства объектов, классифицируемый объект относится к тому классу, которому принадлежат ближайшие к нему объекты обучающей выборки [1].

2.7.3 Классификатор Роше проводит рубрикацию документа исходя из его близости к эталонам рубрик. Эталоном для рубрики с является вектор (w_1, w_2, . . .) в признаковом пространстве, вычисленный по формуле:

$$w_i = \frac{\alpha}{|POS(c)|}\sum_{d \in POS(c)} w_{di} - \frac{\beta}{|NEG(c)|}\sum_{d \in NEG(c)} w_{di}, \quad (2.7)$$

где $POS(c)$ и $NEG(c)$ – множества документов из обучающей выборки, которые принадлежат и не принадлежат рубрике с соответственно, а w_{di} — веса i-го признака документа d.

2.7.4 Метод опорных векторов SV [6]. Основная идея метода – поиск разделяющей гиперплоскости с максимальным зазором в пространстве признаков. Две параллельных гиперплоскости строятся по обеим сторонам гиперплоскости, разделяющей классы. Разделяющей гиперплоскостью будет гиперплоскость, максимизирующая расстояние до двух параллельных гиперплоскостей. Алгоритм работает в предположении, что чем больше разница или расстояние между этими параллельными гиперплоскостями, тем меньше будет средняя ошибка классификатора.

2.7.5 В классификаторах на основе решающих правил пространство данных моделируется набором правил, на левой стороне которого находится условие на набор признаков, а на правой – метка класса. Набор правил генерируется из обучающей выборки. Для данного тестового объекта мы получаем набор правил, для которых он удовлетворяет их условию на левой стороне. Затем определяем метку класса как функцию от полученных меток класса правил. В более общем виде, левая сторона правила является логическим условием, выраженным в виде дизъюнктивной нормальной формы (ДНФ). Тем не менее, в большинстве случаев, условие на левой стороне гораздо проще и представляет собой набор термов, все из которых должны наличествовать в документе для удовлетворения условию.

2.7.6 Вероятностные классификаторы рассматривают решение об отнесении документа d к классу c как вероятность $P(c|d)$ принадлежности этого документа к этому классу, и, соответственно, вычисляют её по теореме Байеса:

$$\mathrm{P(c|d)} = \frac{\mathrm{P(d|c)} * \mathrm{P(c)}}{\mathrm{P(d)}}. \qquad (2.8)$$

2.7.7 Линейные классификаторы. Линейными называют классификаторы, в которых предсказанное значение вычисляется в виде

$$c=w\cdot d+b, \tag{2.9}$$

где $d = (d_1, \ldots, d_l)$ – нормализованный вектор из частот слов в документе;

w – вектор линейных весов той же размерности, что и признаковое пространство;

b – некоторое скалярное значение.

Естественной интерпретацией для c в дискретном случае будет разделяющая гиперплоскость между различными классами.

2.7.8 Нейронные сети. Базовой единицей нейронной сети является нейрон. Каждый нейрон получает набор входов, которые будем обозначать d_i, которые в нашей задаче будут обозначать вхождение термов в i-й документ. Каждый нейрон также ассоциирован с набором весов w, которые используются для подсчета функции входов $f(\cdot)$. Типичной такой функцией, используемой в нейронной сети, является линейная функция:

$$\mathrm{p}_i = \mathrm{w} \cdot d_i \,. \tag{2.10}$$

2.8 Проблемы классификации

Автоматическая классификация текстов весьма сложная задача. В процессе классификации могут возникнуть следующие проблемы.

Первая проблема – предобработка входных данных. Сложность данного этапа состоит и в размере данных – документы содержат десятки тысяч различных слов, количество классов так же может достигать тысячи – и это все при достаточно скудном описании классов (по несколько документов на класс) и небольшом количестве рубрик у каждого документа

(обычно не более 5-8). Также в текстах содержится множество «шума», который не дает нам представления о принадлежности документа к классу. Задача предобработки сводится к вычленению из текста только необходимых, свойственных классу слов. Этого можно достичь путем стирания слова «классической» лексики, слов малой длины (местоимения, предлоги и т.д.) и удаление синонимов и однокоренных слов.

Вторая – выбор метода классификации. На сегодняшний день большинство методов показывают весьма малую точность классификации текстов (~50%).

Также существует проблема ресурсоёмкости. При наличии большого количества классов и ещё большего количества слов внутри классов, требуют весьма большие вычислительные мощности. Каждый из этапов классификации занимает много времени, что в сумме дает весьма долгое время работы даже для одной книги.

И, собственно, имеет место быть невозможность классификации специальных текстов (например, математических, содержащих много формул)

3 Математические формулы в текстовых документах

3.1 Классификация документа с математическими формулами

Рассмотрим последнюю проблему более внимательно. Во множестве научных текстов имеется в среднем от 40% до 80% математических формул, которые представляют собой специальные объекты типа Microsoft Equation в Word или ему подобных. Классические методы классификации способны работать только с текстовой информацией, что делает их неприменимыми для разбора таких особых объектов. Прежде чем разобраться, как решить эту проблему, стоит поподробнее рассмотреть насколько формулы в тексте влияют на результат классификации.

Для этого была сделана выборка из 20 документов математических наук и более 100 документов в сумме по другим жанрам, таким как биология, право, зоология, психология, философия и т.д. Благодаря такому сильному разбросу категорий в выборке, мы сможем практически точно определить, относится ли документ к математике или нет.

Теперь рассмотрим, как же все-таки влияют формулы на качество классификации. Был создан тестовый документ, состоящий только из одних формул, которые по мере проверки заменялись на текстовую информацию. Краткие результаты данного эксперимента приведены в таблице 1.

Таблица 1 – Результаты классификации текстов с формулами

Количество формул в тексте	Результат классификации	Коэффициент *D* схожести для категорий (усредненное значение)			
		Математика	Философия	Биология	Право
100 – 80 %	Категория определена неверно	0,08548	0,008699	0,07745	0,1065
80 – 60 %	Категория определена неверно	0,002876	0,002687	0,001853	0,008699
60 – 40 %	Категория определена верно	1,356	0,8716	0,7514	0,004143
40 – 20 %	Категория определена верно	5,339	3,007	2,478	0,5514
20 – 0 %	Категория определена верно	6,5386	1,045	0,0056	1,503

В качестве коэффициента похожести был выбран простой коэффициент D, который вычисляется по формуле.

$$D = \frac{C}{C_0}, \tag{3.1}$$

где D – коэффициент схожести;

C – количество похожих слов;

C_0 – количество слов в выборке определённой категории.

Количество формул в тексте рассчитывается исходя из того, сколько символов было использовано для описания данных объектов в процентном соотношении. Как мы можем заметить, при достаточно большом количестве формул в тексте (>60%), категория определяется неверно. Это легко объяснить тем, что если мы классифицируем текст с такими объектами, то при преобразовании документа в набор слов формулы просто становятся набором букв или слов, и это только в том случае, если формулы представлены в тексте как особый объект. Если же формулы вставлены как изображения, то они и вовсе не учитываются при классификации. В итоге, когда мы представляем такой тексте в виде набора слов, то он будет либо очень небольшим, либо вообще отсутствовать, что в обоих случаях дает весьма плохие результаты при классификации.

Хотелось бы уточнить, что хоть в таблице с данными и указано, что при интервале в 40%…60 % классификация верна, так можно утверждать только с некоторыми допущениями. Для верного результата в тексте должно быть минимум слов общего лексического значения, иначе при преобразовании набор слов также окажется весьма малым для верной классификации.

В промежутке 0%…40 % можно утвердительно сказать, что формулы не сильно влияют на качество классификации.

Таким образом мы имеем достаточно большое количество документов, которые могут быть неверно классифицированы из-за данной проблемы. Поэтому необходимо найти и применить какие-либо способы для обхода этой проблемы. Вариантами решения могут несколько методов:

- преобразование формул из особых объектов в обычное текстовое представление, для дальнейшей классификации классическими способами;

- нахождение новых алгоритмов классификации, которые бы могли учитывать формульные объекты при определении принадлежности документа.

Рассмотрим более детально первый способ решения проблемы.

3.2. Подходы к классификации текстов, содержащих формулы

В качестве решения проблемы классификации текстов, содержащих формулы можно предложить 2 основных подхода:

- преобразование формул из особых объектов в обычное текстовое представление, для дальнейшей классификации классическими способами;
- нахождение новых алгоритмов классификации, которые бы могли учитывать формульные объекты при определении принадлежности документа.

В работах [3,9] было предложены способы решения проблемы классификации текстов, содержащих формулы в рамках обоих подходов.

Рассмотрим более детально способ решения данной проблемы в рамках первого подхода.

3.3 Преобразование математических формул в текст с помощью Speech-to-Text инструментов

Одним из вариантов приведения формул к нужному виду является их диктовка с последующим преобразованием голоса в текстовую информацию. Крупные корпорации предоставляют инструменты для проведения подобных операций. Их минусом является наличие постоянного подключения к интернету, поскольку преобразование речи происходит на стороне сервера, куда мы отправляем куски звукового файла с нашей речью. На сегодняшний

день имеются два основных движка для такой операции – это Google Speech API и Yandex SpeechKit. Рассмотрим каждый из них более детально.

3.3.1 Google Speech API [10]. Корпорация Google предоставляет за небольшую плату всем желающим свои инструменты для преобразования голоса в текст. Сам механизм основан на нейросети глубокого обучения. Это означает, что чем больше люди пользуются данным сервисом, тем точнее и качественнее получается текст на выходе.

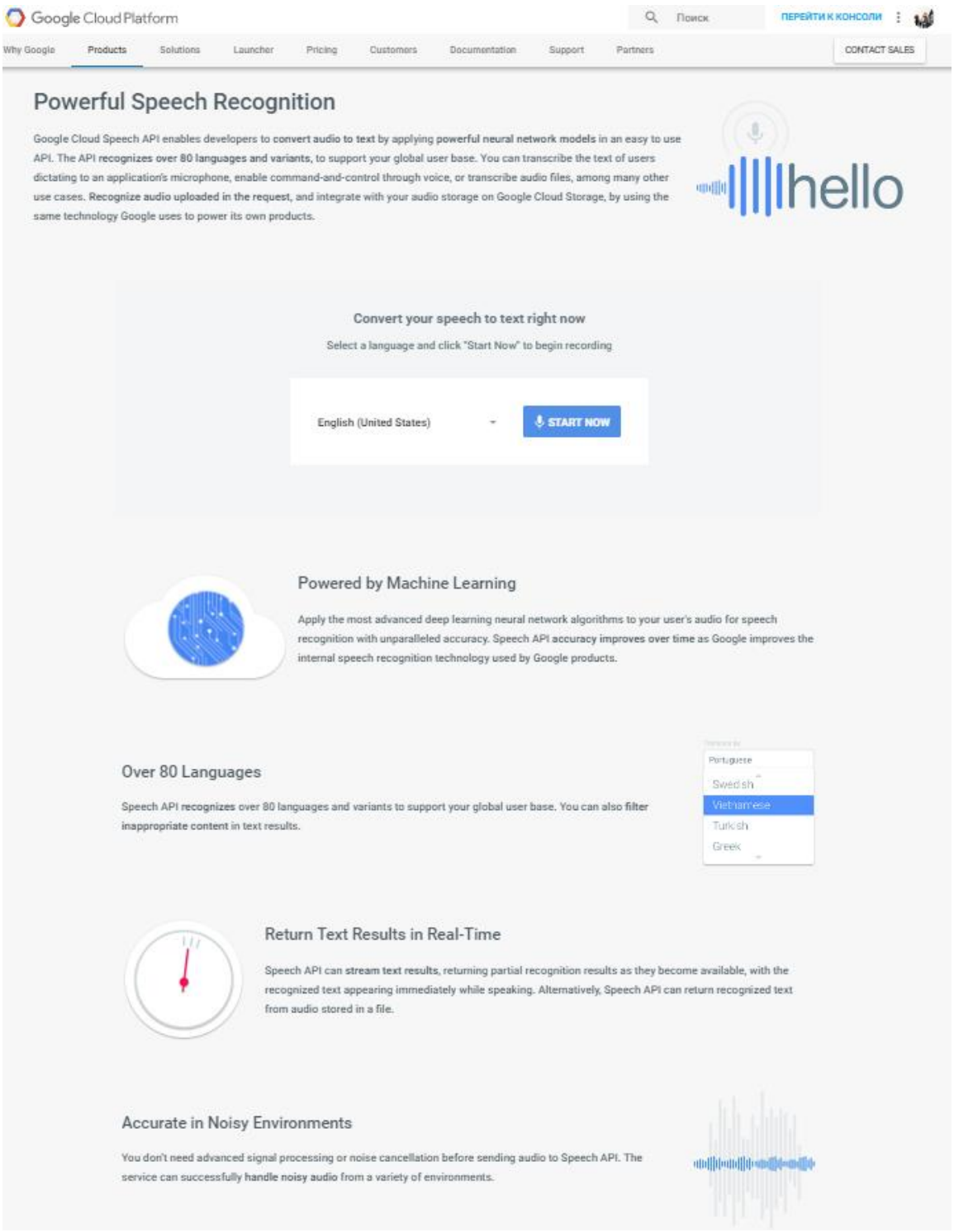

Рисунок 1 – Скриншот интерфейса Google Speech API

Для распознавания доступно более 90 языков, что делает инструмент от Google весьма удобным и универсальным решением в данном вопросе. К сожалению информация по этому сервису для обычного пользователя весьма

общая, нет никаких более точных данных для изучения о способах и принципах распознавания.

Проверим, какой результат может выдать данный движок при диктовке. В качестве примера было взято следующее выражение: $\int_0^R \frac{2xdx}{1+x^2} = \log(1 + R^2)$.

После проговаривания данной формулы и дальнейшей обработки речи, мы получаем текстовое представление:

Интеграл от 0 до 2 X деленное на 1 плюс икс в квадрате икс равно логарифм скобка открывается 1 плюс в квадрате скобка закрывается

Можно заметить, что движку практически не удается распознать короткие выражения, например, такие как: R, dx. Такой недостаток, возможно, не сильно будет влиять на качество классификации, но в особых ситуация такие мелкие пропуски могут сильно повлиять на конечный результат. Также очень важно насколько четко диктуется формула. Для хорошего качества необходимо проговаривать формулу очень медленно, делая паузы между словами. Такое условие весьма сильно увеличивает время на обработку одного текста.

3.3.2 Yandex SpeechKit [12]. В распознавании формул также весьма хорошо преуспела российская кампания Yandex. В отличие от Googlex, Яндекс дает чуть более подробное описание технологи, рассказывая пользователю о языковых моделях и качестве распознавания.

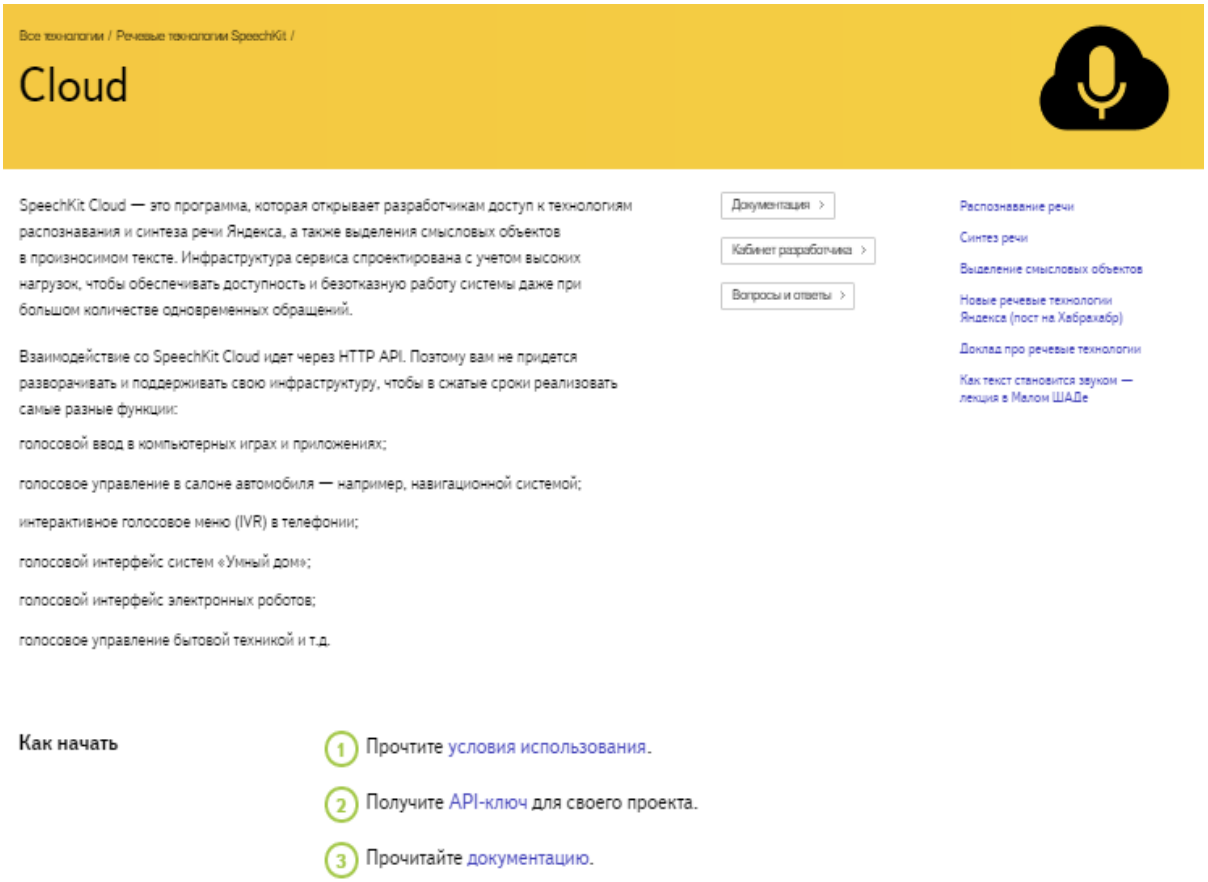

Рисунок 2 – Скриншот интерфейса Yandex SpeechKit

SpeechKit решает задачу распознавания в два этапа. На первом этапе в аудио сигнале выделяются наборы звуков, которые могут быть интерпретированы как слова. Для каждого набора звуков обычно существует несколько вариантов слов – то есть несколько гипотез.

На втором этапе подключается **языковая модель**, которая позволяет проверить каждую гипотезу с точки зрения структуры языка и контекста – насколько данное слово согласуется со словами, распознанными ранее. Система распознавания проверяет гипотезы, пользуясь языковой моделью как словарем. Создание такого словаря – это сложная вычислительная задача, здесь используется машинное обучение нейронных сетей.

Нейронная сеть обучается на речи, которая обычно используется в той или иной области. Поэтому языковые модели специализируются на распознавании речи определенной тематики. Например, для распознавания

номера телефона лучше всего подходит модель **Числа**, а для того чтобы распознать имя и фамилию абонента, следует использовать модель **Имена**.

Точность распознавания зависит от качества исходного звука, качества кодирования аудио, разборчивости и темпа речи, сложности фраз и их длины. Важно, чтобы тематика речи соответствовала выбранной языковой модели — это повышает точность распознавания.

Скорость распознавания зависит от способа передачи звуковых данных. Если данные передаются частями, распознавание происходит одновременно с передачей данных. В этом случае разрыв между окончанием отправки данных и получением результата обычно не превышает 1 секунды.

К минусам SpeechKit можно отнести всего 4 языка для распознавания (русский, английский, турецкий, украинский). Этого количества возможно хватит на начальных этапах разработки программы, но для выхода на международный рынок требуется наличие гораздо большего количества языков.

Рассмотрим качество работы данного движка. Для примера рассмотрим интеграл $\int_0^R \frac{2xdx}{1+x^2} = \log(1+R^2)$.

Результатом работы будет следующий текст:

интеграл от 0 до кр 2 икс деленное на 1 + икс в квадрате под икс равно логарифм скобка открывается 1 + кр в квадрате скобка закрывается

Видно, что в отличие от Google Speech API, движок яндекса намного лучше справляется с распознаванием коротких слов, хотя в одном из случаев неверно определяет «R» как «кр». Но также, как и в инструменте от Google необходимо максимально разборчиво диктовать формулу для хорошего результата.

Описанные выше движки идеально подходят для интеграции системы преобразования речи в текст в программу классификации текстов, а также дают весьма качественные результаты при определенных условиях, но они имеют один существенный минус – платность. Это приведет к сильному удорожанию программы, так как она работает с очень большими объемами текстовой информации.

Если же говорить о бесплатных инструментах, то мы можем посмотреть в сторону продуктов, дающих нам на выходе текст, но без какой-либо интеграции с программой. Это различного рода голосовые блокноты, принцип их работы основан на приведенных выше движках. Самым известным из таких блокнотов является **SpeechPad** [2].

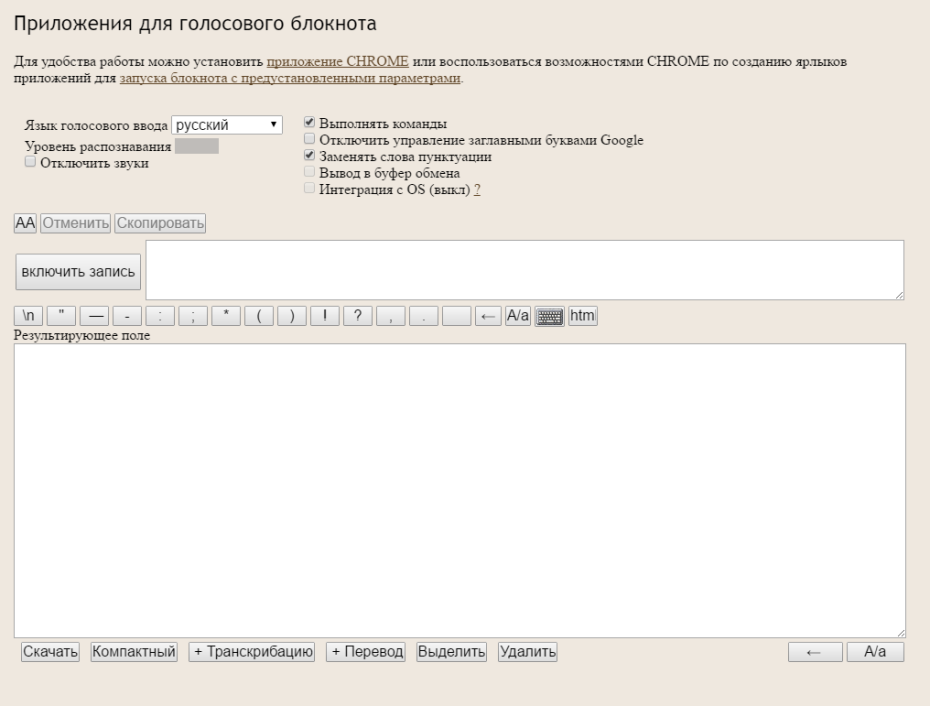

Рисунок 3 – Пример интерфейса голосового блокнота

Одним из наиболее важных факторов, который влияет на работоспособность всего метода, является то, каким образом оператор будет надиктовывать математические формулы. Поскольку каждый человек может по-разному произносить одни и те же выражения, например, вместо «умножить» говорить «на» или не произносить вслух скобки, то необходима

некая стандартизация такой диктовки, чтобы все формулы на выходе имели некий стандартный вид. Поэтому рассмотрим следующий, уже имеющий такую стандартизацию, способ, а именно преобразование в язык верстки TeX.

3.4 Перевод формул в систему верстки TeX

Одним из наиболее важных факторов, который влияет на работоспособность всего метода, является то, каким образом оператор будет надиктовывать математические формулы. Поскольку каждый человек может по-разному произносить одни и те же выражения, например, вместо «умножить» говорить «на» или не произносить вслух скобки, то необходима некая стандартизация такой диктовки, чтобы все формулы на выходе имели некий стандартный вид. Поэтому рассмотрим следующий, уже имеющий такую стандартизацию, способ, а именно преобразование в язык верстки TeX.

TeX – система компьютерной вёрстки, разработанная американским профессором информатики Дональдом Кнутом в целях создания компьютерной типографии. В неё входят средства для секционирования документов, для работы с перекрёстными ссылками. TeX – один из лучших способов для набора сложных математических формул. В частности, благодаря этим возможностям, TeX популярен в академических кругах, особенно среди математиков и физиков.

Например, формула $\int_0^R \frac{2xdx}{1+x^2} = \log(1+R^2)$ в tex-формате будет иметь следующий вид:

\[\int_0^R\frac{2x\,dx}{1+x^2}=\log(1+R^2).\]

При использовании такого подхода необходимо решить две проблемы:

- выделение текстовых блоков в документе;
- выделение формульных блоков в документе.

Формулы, представленные в tex-формате можно добавить к текстовым блокам и воспользоваться известными методами классификации текстовых документов.

Недостатком такого подхода является высокая сложность или невозможность перевода свёрстанного документа (например, pdf или doc файл) в tex-представление. Поэтому для обучения и классификации нужны документы в исходном tex-формате.

Также одним из способов решения выше описанного недостатка является плагин GrindEQ Math для Microsoft Word, который способен преобразовать файлы формата doc в tex-представление.

В качестве примера приведем перевод исходного текста в doc-формате в текст tex-формате.

```
Перевод формул в систему верстки TeX


TeX -- система компьютерной вёрстки, разработанная американским
профессором информатики Дональдом Кнутом в целях создания компьютерной
типографии. В неё входят средства для секционирования документов, для
работы с перекрёстными ссылками. TeX -- один из лучших способов для
набора сложных математических формул. В частности, благодаря этим
возможностям, TeX популярен в академических кругах, особенно среди
математиков и физиков.

Например, формула


\[\int^R_0{\frac{2xdx}{1+x^2}}=\mathrm{log}\mathrm{(}(1+R^2)\]


в tex-формате будет иметь следующий вид:
```

```
{\textbackslash}[{\textbackslash}int\_0$\wedge$R{\textbackslash}frac$\{$2
x{\textbackslash},dx$\}$$\{$1+x$\wedge$2$\}$={\textbackslash}log(1+R$\wed
ge$2).{\textbackslash}]
```

Рисунок 4 – Пример преобразованного в tex текста

Данный плагин имеет бесплатный пробный период, позволяющий приводить к нужному виду лишь небольшие объемы документов. Для полноценной требуется платная регистрация, что ведет к непосредственному удорожанию, но поскольку это лишь один из способов перевода математических формул в TeX, то он не является обязательным к применению. Помимо того плагин на выходе дает много мусора в конечном документе, что не сильно будет влиять на классификацию, если все лишнее внести в список «нормальных» слов, которые вычищаются из текстов при преобразовании их в набор слов.

Замечание: все описанные способы преобразования математических формул стоит рассматривать как дополнительную предобработку текстов перед тем как загружать их в программу классификатор. Таким образом большое количество ручной работы, необходимой в каждом из методов преобразования, можно не учитывать в общем времени классификации текстового документа. Поскольку, в ином случае, данные способы будут весьма сильно замедлять классификацию текста, а иных способов преобразования формул на сегодняшний день практически не существует.

4 Программная реализация для классификации документов, содержащих формулы

В основе прикладной программы для определения категории текстового документа будет использоваться простой метрический алгоритм, основанный на определении похожести слов в текстовом документе и в существующих категориях. Для этого в программе реализована функция сравнения двух слов, возвращающая в результате процент схожести входных слов. Рассмотрим общую структуру ПО.

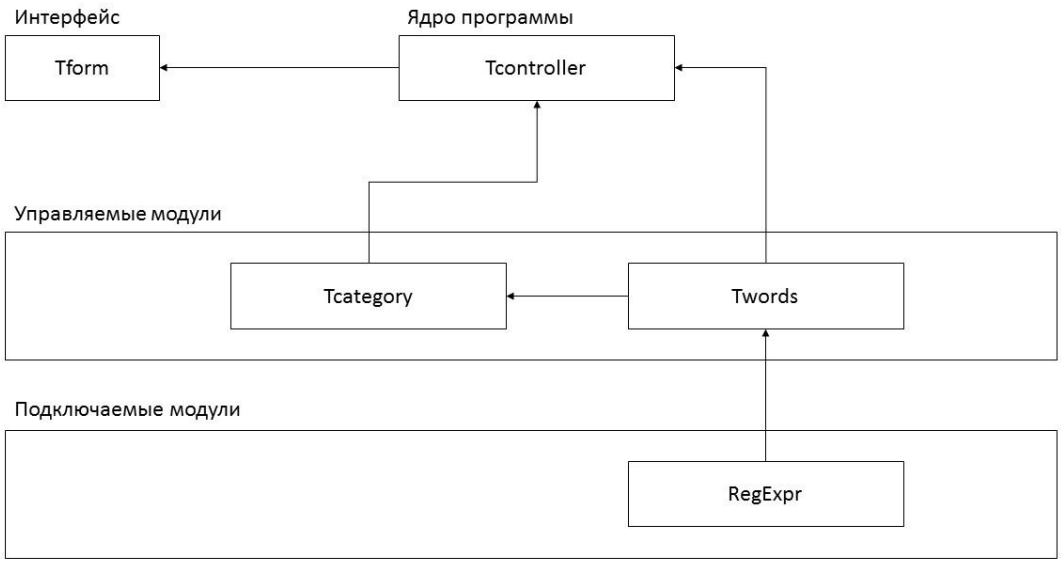

Рисунок 5 – Структурная схема программы

Ядром всей программы является юнит TcontrollerUnit, в котором описан класс Tcontroller. Код класса представлен в приложении В.

Он несет в себе все основные процедуры и функции, которые необходимы для корректной работы. Также в нём содержится массив всех категорий. За каждую отдельную категорию отвечает класс Tcategory (юнит TcategoryUnit). Код класса представлен в приложении В.

В него записываются все необходимые данные, такие как: название, код УДК и список слов, класс которого содержится в юните TwordsUnit. Код класса представлен в приложении В.

Внутри класса Twords содержатся все необходимые методы для выполнения работ по разделения текста на отдельные «термы», очистки дубликатов и создания списка слов для определенной категории. Самой важной процедурой всей программы, которая является реализацией метода классификации текстового документа, является процедура ScanText в юните Tcontroller. Опишем общий принцип работы данной процедуры:

На вход подается текст для определения категории. Он обрабатывается классом Twords, преобразующим текстовую информацию в текстовый список. Далее, перебирая этот список «термов» и сравнивая слова со словами из категорий, содержащих определяющие выборки, создается массив вхождений слов в различные категории.

Опишем принцип работы функции, сравнивающей строки между собой. На вход подаются два слова и минимальный процент схожести. Далее «термы» сравниваются между собой, если они не равны, то убирается один знак в конце каждой из строк и снова сравниваются. Данный процесс будет идти до тех пор, пока слова не станут равны или же процент схожести не опуститься ниже минимальной планки, которую мы передавали одним из параметров. Как вы можете заметить, данный способ практически в лоб сравнивает два слова и является весьма ресурсоемким, но при этом он дает весьма хорошие результаты и поскольку в данной работе не стояло задачи нахождения быстрого и точного метода сравнения слов, а в самом Delphi нет никакой программной реализации подобного, то применение его оправдано.

Итак, мы уже имеем массив чисел для каждой из категорий, которые показывают число схожих слов. Важно заметить, что на данном этапе создается погрешность, из-за которой текст может неверно классифицироваться. Она возникает благодаря тому, что число

определяющих «термов» в выборка для каждой из категорий может очень сильно разниться, например, 2414 слов для категорий «Право» и 32006 для категории «Биология». Такая неравномерность объясняется тем, что некоторые из категорий обладают общим или очень схожим набором специфических слов, а также очень важно какую книгу для обучающей выборки загрузили в программу, т.к. в некоторых на 100 страниц может быть 80 страниц специфических слов, а в других всего 10 или меньше страниц. И при классифицировании для категорий, обладающих большой выборкой, число вхождения может быть выше только потому, что совпало больше слов.

Для устранения данной проблемы число в каждой позиции массива необходимо разделить на количество определяющих слов соответствующей категории. Тогда мы усредним результаты и получим некий «коэффициент похожести». Для обобщения запишем формулу вычисления этого числа:

В итоге находя максимальное число в выходном массиве мы получаем данные о категории, к которой может принадлежать классифицируемый документ. В результатах нет 100% точности, поэтому мы говорим, что текст лишь может принадлежать. Мы можем говорить о том, что тексты, состоящие более чем на 60% из таких объектов, которые прошли специальную предобработку, будут весьма хорошо классифицироваться, кроме исключений, которые без сомнения могут возникать в процессе работы, т.к. количество текстов исчисляется миллионами.

5 Результаты классификации для методов преобразования математических формул

Ранее мы уже показали, насколько сильно формулы могут влиять на качество классификации текстового документа. Рассмотрим, как повлияли на конечный результат методы преобразования формул. Мы выделили два основных метода. В дискретной модели – это преобразование в формат TeX, а в непрерывной модели перевод текстовой информации в звуковой сигнал с дальнейшим превращением его обратно в текст с помощью специальных инструментов. Для рассмотрения работы этих методов сделана выборка из 400 документов с заранее известными категориями: математика, биология, философия, право – по 100 документов на каждую категорию. Был взят тестовый документ, на 60% состоящий из формул, который в результате должен быть отнесен к категории «математика». Процентное соотношение формул ко всему тексту, после преобразования каждым из способов представлено в таблице 2.

Таблица 2 – Процентное соотношение формул ко всему текстовому документу

Число формул в тексте до преобразования	Число формул в тексте после преобразования	
	Перевод в TeX формат	Speech-To-Text
80%	~82%	~87%
60%	~61%	~69%
40%	~42%	~46%
20%	~21%	~27%

Это соотношение вычисляется количеством символом формульных выражений к общему количеству символов в тексте. Как можно заметить при переводе в TeX формат процент не слишком увеличивается, поскольку

формулы переходят от обычного вида к линейному с некоторыми преобразованиями, которые не сильно увеличивают длину выражения. Важно заметить, что при использовании этого способа появляется некоторый мусор, который очищается на этапе предобработки, поэтому мы не учитываем его при вычислении соотношения. В Speech-To-Text процент достаточно сильно увеличивается, поскольку в текстовом представлении формульное выражение имеет более длинную запись, в отличие от того, если бы мы писали формулу с помощью специальных символов. Но нужно заметить, что при таком представлении задача классификации становится более легкой, так как весь текст теперь состоит из обычных слов.

При изменении количества формул в тексте разница между оригинальным текстом и преобразованным будет всегда примерна одинаковой, ~1% - ~ 3% для метода перевода в TeX и ~6 - ~9% для Speech-To-Text. Процент может варьироваться, так как разные формульные символы имеют разную длину в представлениях, и их наличие или отсутствие может увеличивать или уменьшать процентное соотношение.

Теперь, когда мы имеем преобразованные документы для каждого из способов, проверим, насколько улучшилось качество классификации. Результаты классификации по обоим методам представлены в таблицах 3 и 4.

Таблица 3 – Результаты классификации с использованием метода перевода в TeX

Количество формул в тексте	Результат классификации	Коэффициент *D* схожести для категорий			
		Математика	Философия	Биология	Право
100 – 80 %	Категория определена неверно	0,9235	0,9356	0,07624	0,0542
80 – 60 %	Категория определена верно	2,0027	1, 2687	1, 853	0,8349
60 – 40 %	Категория определена верно	3,635	1,5746	1,7152	0,452
40 – 20 %	Категория определена верно	6,6835	4,257	2,387	1,5158
20 – 0 %	Категория определена верно	7,8356	3,054	0,56	1,801

Таблица 4 – Результаты классификации с использованием метода Speech-To-Text

Количество формул в тексте	Результат классификации	Коэффициент *D* схожести для категорий			
		Математика	Философия	Биология	Право
100 – 80 %	Категория определена верно	1,8838	0,7479	0,2091	0,0292
80 – 60 %	Категория определена верно	2,73	1,1069	1,0749	0,5731
60 – 40 %	Категория определена верно	5,4363	1,7219	2,3329	1,2035
40 – 20 %	Категория определена верно	6,9817	4,8547	3,0244	4,0082
20 – 0 %	Категория определена верно	9,3547	4,5856	3,5056	2,6571

Коэффициент *D* вычисляется по формуле (3.1).

Можно заметить, что при объеме формул в тексте от 80% до 100% способ перевода звукового сигнала работает гораздо лучше, чем перевод в TeX. Важно отметить, что преобразования файле в Tex представление все-таки может давать верный результат классификации, но лишь в малом числе случаев. При объеме формул менее 80% оба метода достаточно хорошо справляются со своей задачей, увеличивая количество возможно классифицируемых документов в несколько раз.

ЗАКЛЮЧЕНИЕ

На сегодняшний день готовых методов для классификации документов с особыми объектами, такими как математические формулы, нет.

Предложенные способы стоит рассматривать как предобработку перед самой классификацией, поскольку в процессе преобразования формул в обоих методах необходимо человеческое вмешательство. В конечном итоге они сводят документы к дискретной модели, поэтому для данных подходов можно использовать известные методы классификации.

В ходе проверки двух методов было установлено, что способ преобразования голоса в текст, при объеме формул в тексте от 80 до 100 процентов, дает более точные результаты, в отличии от метода перевода текстового документа в TeX формат. Поэтому можно утверждать, что метод приведения непрерывного сигнала к дискретной модели более универсален и эффективен. Но нужно заметить, что его эффективность сильно зависит от того, насколько четко оператор будет проговаривать формулы, а также насколько движок может понимать предлагаемый звуковой сигнал для перевода в текст.

На сегодняшний день все инструменты Speech-To-Text предъявляют достаточно жёсткие требования к произношению текста.

При достаточно хорошей реализации и оптимизации предложенных методов, число текстовых документов, содержащих формулы, может быть значительно увеличено, что существенно повысит качество классификации. Учитывая полученные результаты, данная работа может дополнить арсенал методов, используемых для классификации текстовых документов.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Вапник, В. Н. Теория распознавания образов : учебное пособие / В. Н. Вапник, А. Я. Червоненкис. – Москва : Наука, 1974. – 416 с.

2. Голосовой блокнот SpeechPad [Электронный ресурс] : движок для преобразования голоса в текстовую информацию – Режим доступа: https://www.speechpad.ru.

3. Лавренов, А. О. Классификация текстов, содержащих формулы / А. О. Лавренов, Б. В. Олейников. – Москва : Изд. Московский университет им. С.Ю. Витте, 2014. – 120 с. - ISSN 2312-5500.

4. Олейников, Б. В. Обобщенный коэффициент подобия для биоценотических исследований / Б. В. Олейников ; КрГУ. – Красноярск, 1984. – 23 с. - Деп. в ВИНИТИ 13.12.84, № 7978–84.

5. Харин, Н. П. Метод ранжирования выдачи, учитывающий автоматически построенные ассоциативные отношения между терминами : учебное пособие / Н. П. Харин. – Москва : НТИ, 1990. – 50 с.

6. Шабанов, В. И. Алгоритм формирования ассоциативных связей и его применение в поисковых системах : учебное пособие / В. И. Шабанов, А. Е. Власова. – Москва : Диалог-2003, 2003. – 700 с.

7. Joachims, T. Making large-scale SVM learning practical : учебное пособие / T. Joachims. – Cambridge : MIT Press, 1999. – 100 с.

8. Manning, C. Introduction to information Retrieval [Электронный ресурс] : онлайн издание / C. Manning, P. Raghavan, H. Schütze. – Режим доступа: https://nlp.stanford.edu/IR-book.

9. Oleynikov, B. V. Text classification based on their audio converting : учебное пособие / B. V. Oleynikov, A. O. Lavrenov. – Moscow : SIIT&T Informika, 2013. – 56 с.

10. Google Speech API [Электронный ресурс] : перевод звука в текст – Режим доступа: https://cloud.google.com/speech/.

11. Sebastiani, F. Machine Learning in Automated Text Categorization : учебное пособие / F. Sebastiani. – Cambridge : ACM Computing Surveys, 2002. – 47 с.

12. Yandex SpeechKit [Электронный ресурс] : преобразование речи в текст – Режим доступа: https://tech.yandex.ru/speechkit/.

ПРИЛОЖЕНИЕ А

(обязательное)

Тестовый текстовый документ до преобразования в набор слов

Описанные в настоящей работе методики были использованы при аппроксимации справочных данных по коэффициентам концентрации упругих напряжений (ККН) для ряда концентраторов, представленных в [5]. В качестве примера с помощью таблицы и рис.3 показан вывод формулы вычисления коэффициента концентрации местных напряжений для зоны галтельного сопряжения цилиндра с плоским днищем при действии внутреннего давления. Указанный ККН (Кσ) зависит от безразмерных переменных X_1 и X_2 , в которые входят геометрические размеры рассматриваемой зоны концентрации.

Окончательная аналитическая зависимость для коэффициента концентрации напряжений имеет вид :

$$K\sigma = \frac{0{,}7}{\sqrt{X_1 \sqrt{X_2}}} + 0{,}8 \ .$$

Данная формула приведена в [5] и в работе [6] .

Другая схожая мера, часто используемая в отборе признаков для документов – это прирост информации. Пусть P_i будет априорной вероятностью принадлежности к классу i, а $p_i(w)$ будет условной вероятностью принадлежности к классу i при условии того, что документ содержит слово w. Пусть $F(w)$ – часть документов, содержащая слово w. Тогда мера прироста информации $I(w)$ для данного слова w определяется как

$$I(w) = -\sum_{i=1}^{k} P_i * \log(P_i) + F(w) * \sum_{i=1}^{k} p_i(w) * \log\big(p_i(w)\big) + \quad (2.4)$$
$$+\big(1 - F(w)\big) * \sum_{i=1}^{k}\big(1 - p_i(w)\big) * \log(1 - p_i(w)).$$

Чем больше значение прироста информации $I(w)$, тем сильнее разделяющая способность слова w.

Статистика хи-квадрат (χ^2) – это еще один способ выявить зависимость (а точнее, отсутствие независимости) между словом w и определенным классом i. Как и ранее, m будет общим количеством документов в выборке, $p_i(w)$ – условная вероятность принадлежности документа к классу i при наличии слова w, P_i – априорная вероятность принадлежности к классу i, $F(w)$ – часть документов, содержащих слово w.

$$\chi^2(w) = \frac{n*F(w)^2*(p_i(w)-P_i)^2}{F(w)*\big(1-F(w)\big)*P_i*(1-P_i)}. \quad (2.5)$$

Наивный байесовский алгоритм – это простой вероятностный классификатор, основанный на применении Теоремы Байеса со строгими предположениями о независимости [1]. Класс принадлежности c документа d считается по формуле:

$$c = argmax_{o\in c} \log\left(\frac{D_o}{D}\right) + \sum_{i\in Q} \log\left(\frac{W_{ic}+1}{n+L_c}\right), \quad (2.6)$$

где C – множество классов;

D_o – количество документов в обучающей выборке принадлежащих классу c;

D – общее количество документов в обучающей выборке;

L_c – суммарное количество слов в документах класса c в обучающей выборке;

W_{ic} – сколько раз i-ое слово встречалось в документах класса c в обучающей выборке;

Q – множество слов классифицируемого документа.

$$w_i = \frac{\alpha}{|POS(c)|}\sum_{d\in POS(c)} w_{di} - \frac{\beta}{|NEG(c)|}\sum_{d\in NEG(c)} w_{di},$$

$$M_i(w) = \log\left(\frac{F(w)*p_i(w)}{F(w)*P_i}\right) = \log\left(\frac{p_i(w)}{P_i}\right).$$

$$I(w) = -\sum_{i=1}^{k} P_i * \log(P_i) + F(w)$$
$$* \sum_{i=1}^{k} p_i(w) * \log\big(p_i(w)\big) + +\big(1 - F(w)\big)$$
$$* \sum_{i=1}^{k} \big(1 - p_i(w)\big) * \log(1 - p_i(w))$$
$$\chi^2(w) = \frac{n * F(w)^2 * (p_i(w) - P_i)^2}{F(w) * \big(1 - F(w)\big) * P_i * (1 - P_i)}$$
$$r(\bar{a}, \overline{b}) = \frac{R*(n_{ij}+a*n_{ij})}{R*(n_{ij}+a*n_{ij})+n_{ij}+n_{ij}};$$

ПРИЛОЖЕНИЕ Б

(обязательное)

Тестовый текстовый документ после преобразования в набор слов с применением фильтрации (без обработки формул)

Log, алгоритм, аналитическая, аппроксимации, априорная, байеса, байесовский, безразмерных, больше, будет, вероятность, вид, внутреннего, встречалось, входят, выборке, вывод, вычисления, выявить, галтельного, геометрические, давления, данная, действии, днищем, документ, другая, зависимость, зависит, значение, зоны, имеет, информации, использованы, используемая, качестве, квадрат, ккн, класс, классификатор, классифицируемого, количество, концентраторов, концентрации, которые, коэффициента, мера, местных, методики, множество, наивный, наличии, напряжений, настоящей, независимости, обучающей, общее, общим, один, окончательная, описанные, определенным, определяется, основанный, отборе, отсутствие, переменных, плоским, показан, помощь, предположениями, представленных, приведена, признаков, применении, примера, принадлежащих, принадлежности, прирост, прироста, простой, пусть, работе, раз, разделяющая, размеры, ранее, рассматриваемой, рис, ряда, сильнее, сколько, слов, содержащая, содержащих, содержит, сопряжения, способ, способность, справочных, статистика, строгими, суммарное, схожая, считается, таблицы, теоремы, тогда, того, точнее, указанный, упругих, условии, условная, формула, формуле, формулы, цилиндра, часто, часть, чем.

ПРИЛОЖЕНИЕ В

(обязательное)

Код структур классов

Tcontroller

```
Tcontroller = class
  settings: _settings; // настройки программы
  normalWords: Twords; // класс слов общей лексики
  categories: array of Tcategory; // массив классов категорий
  constructor create;
  procedure setNormalWords; // добавить слова общей лексики
  procedure clearNormalWordsOfClasses(index: integer); //очистка слов общей лексики из выборки  определенной категории
  procedure addCategory(_name: string; _UDKcode: integer); //добавить категорию
  procedure deleteCategory(index: integer); //удалить категорию
  procedure deleteWordsInCategory(index:integer); //удалить выборку из категории
  procedure addWordsToCategory(index: integer); //добавить выборку в категорию
  function openDialog: string;
  procedure loadSettings; // загрузка настроек
  procedure saveSettings; //сохранение настроек
  procedure saveSession; // сохранение сессии
  function CompareWords(word1: string; word2: string; minpercent: real): real; // сравнение двух слов на похожесть
  procedure removeNormalWords(index: integer);
  procedure ScanText(memo:Tmemo); // определить категорию для текста
end;
```

Tcategory

```
type
Tcategory = class
  name:string[255]; //название категории
  UDKcode:integer; // УДК код категории
  wordsList:Twords; // выборка категории
  constructor create(_name:string;_UDKcode:integer;_wordsList: Twords);
end;
```

Twords

```
  Twords = class
    minLengthWords: integer;
    list: array of _words;
    procedure openFile(path: string);
    procedure addWords(str: string); // добавить слова в выборку
    procedure deleteWord(index: integer); //удалить слово из выборки
    procedure RemoveDuplicates; // удалить повторяющиеся слова
    procedure Clear; //удалить все слова из выборки
  end;
```

ПРИЛОЖЕНИЕ Г

(обязательное)

Скриншоты интерфейса прикладного ПО

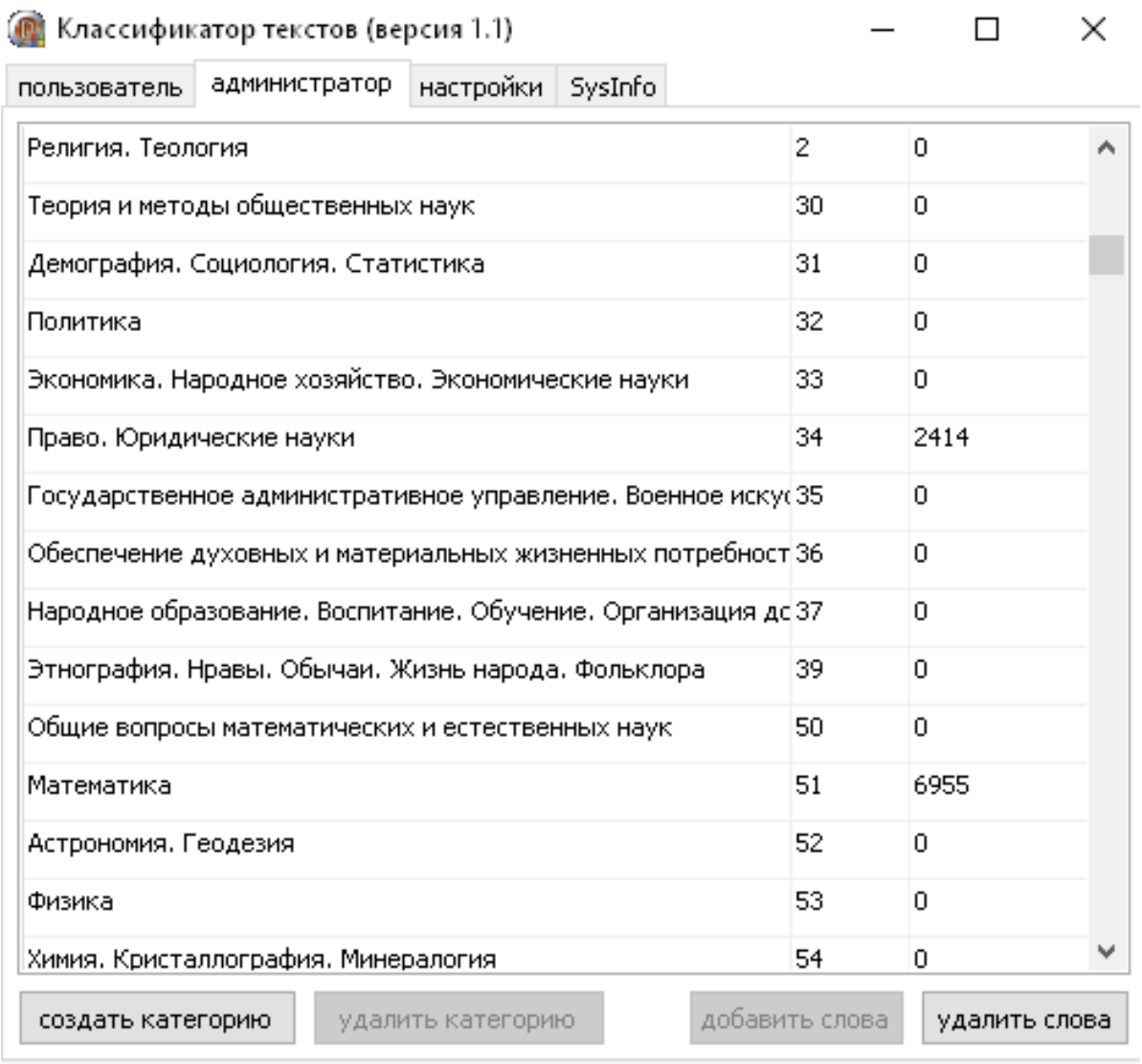

Рисунок Г. 1 – панель администратора.

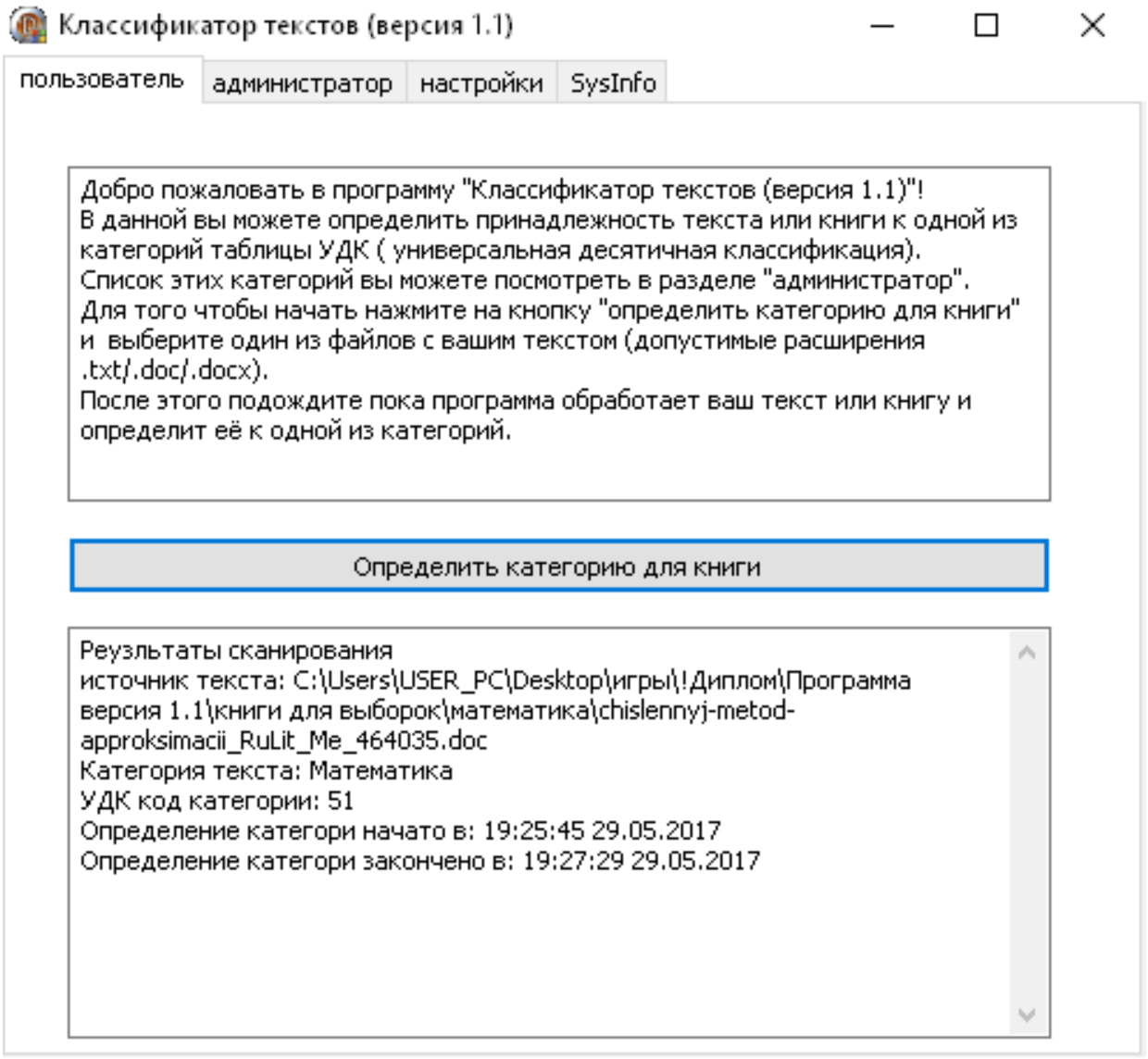

Рисунок Г. 2 – панель пользователя.

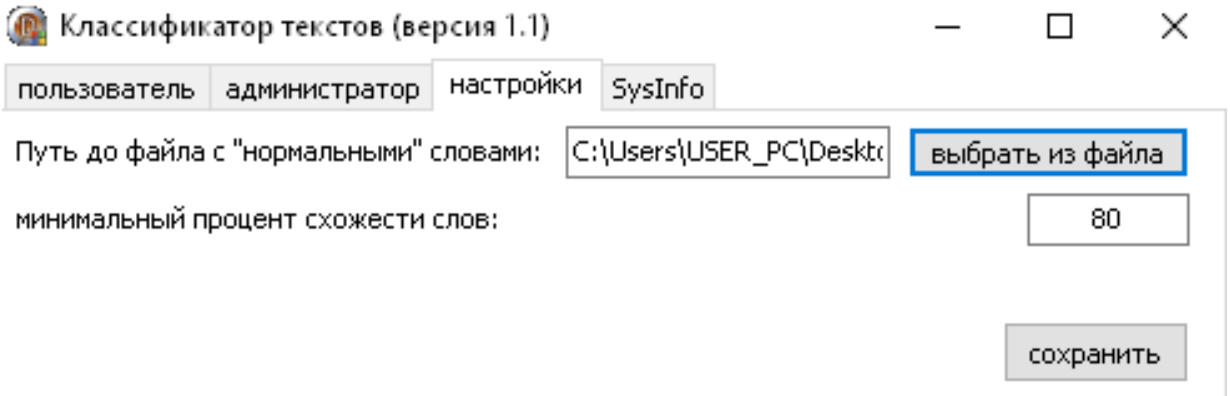

Рисунок Г. 3 – настройки.

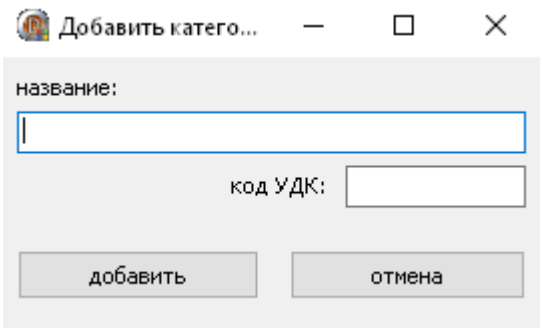

Рисунок Г. 5 – окно добавления новой категории.

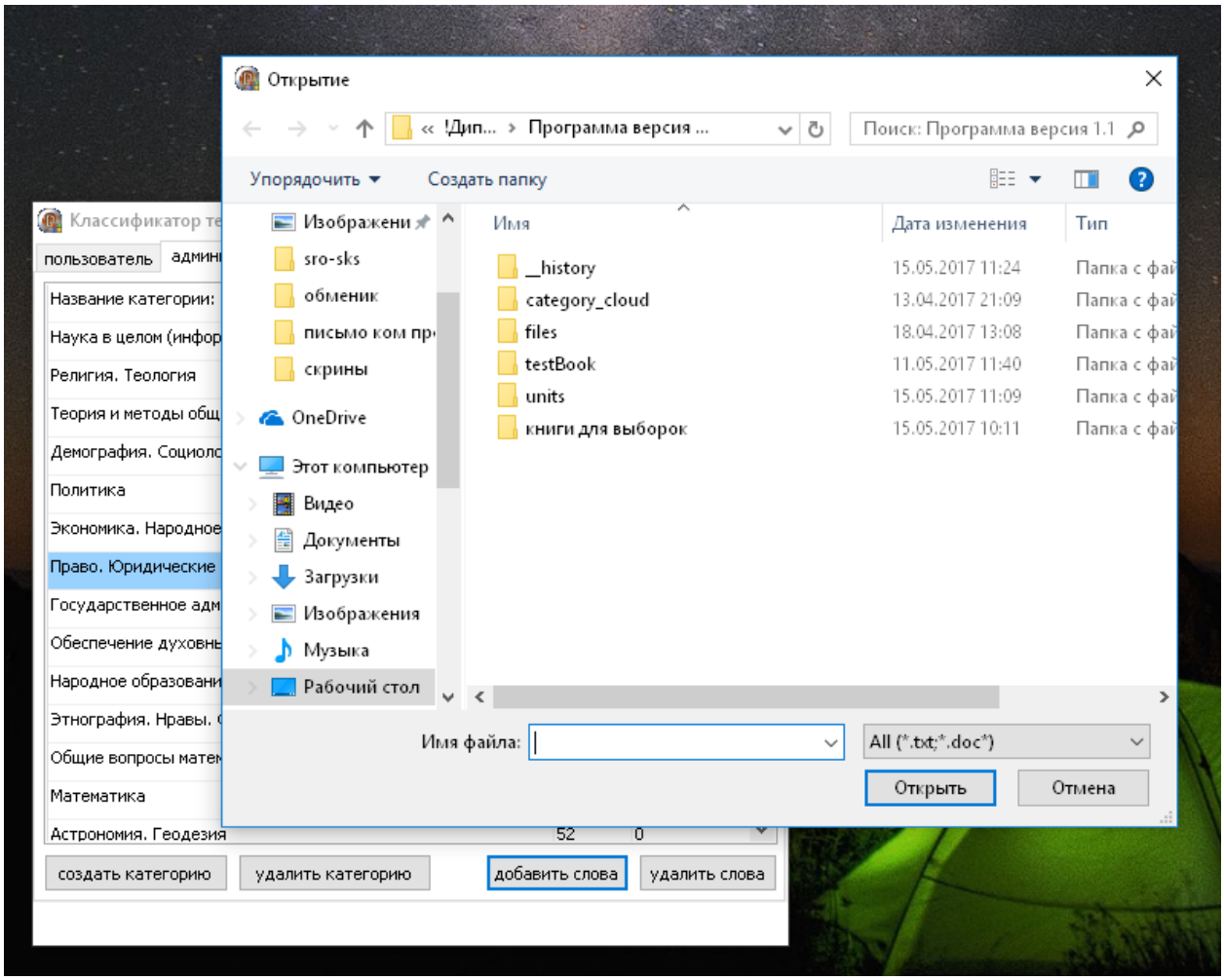

Рисунок Г. 6 - окно добавления новых слов для выбранной категории

ПРИЛОЖЕНИЕ Д

(обязательное)

Код прикладного ПО для классификации текстов с формулами

Юнит BookClass.pas

```
unit bookClass;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, TcontrollerUnit, addCategoryDialog, StdCtrls, Grids, ComCtrls;

var
StatusLabelGlobal:Tlabel;
StatusProgressBarGlobal:Tprogressbar;

type
  TForm1 = class(TForm)
    PageControl: TPageControl;
    user: TTabSheet;
    system: TTabSheet;
    settings: TTabSheet;
    categoriesTable: TStringGrid;
    addCategoryButton: TButton;
    deleteCategoryButton: TButton;
    addWordsToCategory: TButton;
    Label1: TLabel;
    normalWordsPath: TEdit;
    comparePercent: TEdit;
    Label2: TLabel;
    saveSettigns: TButton;
    Button1: TButton;
    ScanText: TButton;
    Memo1: TMemo;
    ResultMemo: TMemo;
    StatusLabel: TLabel;
    StatusProgressBar: TProgressBar;
    TabSheet1: TTabSheet;
    SysInfo: TMemo;
    DeleteWords: TButton;
    procedure addCategoryButtonClick(Sender: TObject);
    procedure FormCreate(Sender: TObject);
    procedure categoriesTableSelectCell(Sender: TObject; ACol, ARow: Integer;
      var CanSelect: Boolean);
    procedure FormDestroy(Sender: TObject);
    procedure saveSettignsClick(Sender: TObject);
    procedure deleteCategoryButtonClick(Sender: TObject);
    procedure Button1Click(Sender: TObject);
```

```
    procedure addWordsToCategoryClick(Sender: TObject);
    procedure ScanTextClick(Sender: TObject);
    procedure DeleteWordsClick(Sender: TObject);
  private
    nowPos: Integer;

    { Private declarations }
  public
    controller: Tcontroller;
    procedure updateGrid;
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.dfm}

procedure TForm1.addCategoryButtonClick(Sender: TObject);
var
  addCategoryDialog: TForm2;
begin
  addCategoryDialog := TForm2.Create(nil);
  addCategoryDialog.Show;
end;

procedure TForm1.addWordsToCategoryClick(Sender: TObject);
begin
controller.addWordsToCategory(self.nowPos);
self.updateGrid;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
self.normalWordsPath.Text:= controller.openDialog;
end;

procedure TForm1.categoriesTableSelectCell(Sender: TObject;
  ACol, ARow: Integer; var CanSelect: Boolean);
begin
  if (ACol < categoriesTable.ColCount) and (ARow < categoriesTable.Rowcount)
    then
  Begin
    nowPos := ARow - 1;

    if nowPos > -1 then
     Begin
       self.deleteCategoryButton.Enabled:=true;
       self.addWordsToCategory.Enabled:=true;
     End
    else
     Begin
       self.deleteCategoryButton.Enabled:=false;
```

```
      self.addWordsToCategory.Enabled:=false;
    End;

  End
end;

procedure TForm1.deleteCategoryButtonClick(Sender: TObject);
var
  i: Integer;
begin
  i := MessageDlg('вы точно хотите удалить категорию: "' + controller.categories
     [nowPos].name + '" ?', mtConfirmation, [mbYes, mbNo], 0);
  if (i = mrYes) and (nowPos > -1) then
  Begin
    controller.deleteCategory(nowPos);
    self.updateGrid;
    showMessage('категория удалена!');
  End;
end;

procedure TForm1.DeleteWordsClick(Sender: TObject);
var
i:integer;
begin
 i := MessageDlg('вы точно хотите удалить слова категории: "' + controller.categories
     [nowPos].name + '" ?', mtConfirmation, [mbYes, mbNo], 0);
  if (i = mrYes) and (nowPos > -1) then
  Begin
    controller.deleteWordsInCategory(nowPos);
    self.updateGrid;
    showMessage('слова категории удалены!');
  End;
end;

procedure TForm1.FormCreate(Sender: TObject);
var
  i: Integer;
begin
  StatusLabelGlobal := self.StatusLabel;
  StatusLabelGlobal.Caption := '';

  StatusProgressBarGlobal := self.StatusProgressBar;

  controller := Tcontroller.Create;
  self.normalWordsPath.Text := controller.settings.normalWordsPath;
  self.comparePercent.Text := inttostr(controller.settings.comparePercent);
  with categoriesTable do
  Begin
    Options := Options - [goEditing];
    ColWidths[0] := 0;
    ColWidths[2] := 50;
    ColWidths[3] := 100;
    ColWidths[1] := width - 150;
    Cells[0, 0] := '№';
    Cells[1, 0] := 'Название категории:';
```

```
        Cells[2, 0] := 'код УДК:';
        Cells[3, 0] := 'количество слов:';
        Rowcount := 1;
      End;
      for i := 0 to length(controller.categories) - 1 do
      Begin
        with categoriesTable do
        Begin
          Rowcount := Rowcount + 1;
          Cells[0, Rowcount - 1] := inttostr(Rowcount - 1);
          Cells[1, Rowcount - 1] := controller.categories[i].name;
          Cells[2, Rowcount - 1] := inttostr(controller.categories[i].UDKcode);
          Cells[3, Rowcount - 1] := inttostr(length(controller.categories[i].wordsList.list));
        End;
      End;
    end;

    procedure TForm1.FormDestroy(Sender: TObject);
    begin
      controller.saveSession;
      controller.Free;
    end;

    procedure TForm1.saveSettignsClick(Sender: TObject);
    begin
      controller.settings.normalWordsPath := self.normalWordsPath.Text;
      controller.settings.comparePercent := strtoint(self.comparePercent.Text);
      controller.saveSettings;
      if controller.settings.normalWordsPath <> self.normalWordsPath.Text then
controller.setNormalWords;
      showMessage('настройки сохранены!');
    end;

    procedure TForm1.ScanTextClick(Sender: TObject);
    begin
    controller.ScanText(self.ResultMemo);
    end;

    procedure TForm1.updateGrid;
    var
      i: Integer;
    begin
      with categoriesTable do
      Begin
         for i := 1 to Rowcount - 1 do
          Rows[i].Clear;
        Rowcount := 1;
        Options := Options - [goEditing];
        ColWidths[0] := 0;
        ColWidths[2] := 50;
        ColWidths[3] := 100;
        ColWidths[1] := width - 150;
        Cells[0, 0] := '№';
        Cells[1, 0] := 'Название категории:';
        Cells[2, 0] := 'код УДК:';
```

```
    Cells[3, 0] := 'колчисетво слов:';
    Rowcount := 1;
  End;
  for i := 0 to length(controller.categories) - 1 do
  Begin
    with categoriesTable do
    Begin
      Rowcount := Rowcount + 1;
      Cells[0, Rowcount - 1] := inttostr(Rowcount - 1);
      Cells[1, Rowcount - 1] := controller.categories[i].name;
      Cells[2, Rowcount - 1] := inttostr(controller.categories[i].UDKcode);
      Cells[3, Rowcount - 1] := inttostr(length(controller.categories[i].wordsList.list));
    End;
  End;
end;

end.
```

Юнит TcontrollerUnit.pas

```
unit TcontrollerUnit;

interface

uses
  SysUtils,
  Classes,
  Dialogs,
  ComObj,
  regexpr, Grids, StdCtrls, TwordsUnit, TcategoryUnit;

type
  _words = record
    word: string[255];
    weight: integer;
  end;

  _settings = record
    comparePercent: integer;
    normalWordsPath: string[255];
  end;

  _cats = record
    name: string[255];
    UDKcode: integer;
  end;

  Tcontroller = class
    settings: _settings;
    normalWords: Twords;
    categories: array of Tcategory;
    constructor create;
    procedure setNormalWords;
    procedure clearNormalWordsOfClasses(index: integer);
    procedure addCategory(_name: string; _UDKcode: integer);
```

```
    procedure deleteCategory(index: integer);
    procedure deleteWordsInCategory(index:integer);
    procedure addWordsToCategory(index: integer);
    function openDialog: string;
    procedure loadSettings;
    procedure saveSettings;
    procedure saveSession;
    function CompareWords(word1: string; word2: string; minpercent: real): real;
    procedure removeNormalWords(index: integer);
    procedure ScanText(memo:Tmemo);
  end;

implementation

{ Tcontroller }

uses
bookClass;

constructor Tcontroller.create;
var
  f: textfile;
  f1: file of _cats;
  f2: file of _words;
  cats: _cats;
  i: integer;
  s: string;
  words: _words;
begin
  self.normalWords := Twords.create;

  loadSettings;

  if FileExists('files/normalWords.dat') then
  Begin
    assignFile(f2, 'files/normalWords.dat');
    reset(f2);

    while not eof(f2) do
    Begin
      read(f2, words);
      setlength(self.normalWords.list, length(self.normalWords.list) + 1);
      self.normalWords.list[length(self.normalWords.list) - 1].word :=
        words.word;
      self.normalWords.list[length(self.normalWords.list) - 1].weight :=
        words.weight;
    End;
    closeFile(f2);
  End;

  if FileExists('files/categoriesList.dat') then
  Begin
    assignFile(f, 'files/categoriesList.dat');
    reset(f);
    while not eof(f) do
```

```
    Begin
      readln(f, s);
      assignFile(f1, 'files/' + s + '.dat');
      reset(f1);
      while not eof(f1) do
        read(f1, cats);
      closeFile(f1);
      setlength(categories, length(categories) + 1);
      categories[length(categories) - 1] := Tcategory.create(cats.name,
        cats.UDKcode, nil);
      assignFile(f2, 'files/' + s + 'Words.dat');
      reset(f2);
      while not eof(f2) do
      Begin
        read(f2, words);
        setlength(categories[length(categories) - 1].wordsList.list,
          length(categories[length(categories) - 1].wordsList.list) + 1);
        categories[length(categories) - 1].wordsList.list
          [length(categories[length(categories) - 1].wordsList.list) - 1]
          .word := words.word;
        categories[length(categories) - 1].wordsList.list
          [length(categories[length(categories) - 1].wordsList.list) - 1]
          .weight := words.weight;
      End;
      closeFile(f2);
    End;
    close(f);
  End;
end;

procedure Tcontroller.addWordsToCategory(index: integer);
var
  path: string;
begin
  path := self.openDialog;
  if path <> '' then
  Begin
    categories[index].wordsList.openFile(path);
  self.clearNormalWordsOfClasses(index);
  showMessage('слова добавлены! (количество: ' + inttostr
      (length(categories[index].wordsList.list)) + ' )');
  End;
end;

procedure Tcontroller.clearNormalWordsOfClasses(index: integer);
var
  i, j: integer;
begin
  for i := 0 to length(self.categories[index].wordsList.list) - 1 do
    for j := 0 to length(self.normalWords.list) - 1 do
      if self.CompareWords(self.categories[index].wordsList.list[i].word,
        self.normalWords.list[j].word, self.settings.comparePercent) > -1 then
        self.categories[index].wordsList.deleteWord(i);
end;
```

```
procedure Tcontroller.addCategory(_name: string; _UDKcode: integer);
var
  i: integer;
begin
  setlength(categories, length(categories) + 1);
  categories[length(categories) - 1] := Tcategory.create(_name, _UDKcode, nil);
end;

procedure Tcontroller.deleteCategory(index: integer);
var
  i: integer;
begin
  DeleteFile('files/' + categories[index].name + '.dat');
  DeleteFile('files/' + categories[index].name + 'Words.dat');
  for i := index to length(categories) - 2 do
    categories[i] := categories[i + 1];
  categories[length(categories) - 1].Free;
  setlength(categories, length(categories) - 1);

end;

procedure Tcontroller.deleteWordsInCategory(index: Integer);
var
f2: file of _words;
begin
 assign(f2,'files/' + categories[index].name + 'Words.dat');
 rewrite(f2);
 setlength(categories[index].wordsList.list,0);
end;

function Tcontroller.openDialog: string;
var
  dialog: TOpenDialog;
begin
  dialog := TOpenDialog.create(nil);
  dialog.InitialDir := GetCurrentDir;
  dialog.Options := [ofFileMustExist];
  dialog.Filter := 'All|*.txt;*.doc*|Text files|*.txt|Word files|*.doc*';
  dialog.FilterIndex := 1;
  if dialog.Execute then
  Begin
    result := dialog.FileName;
    dialog.Free;
  End
  else
  Begin
    showMessage('Открытие файла остановлено');
    result := '';
    dialog.Free;
  End;

end;

procedure Tcontroller.saveSession;
var
```

```
  i, j: integer;
  f: textfile;
  f1: file of _cats;
  f2: file of _words;
  cats: _cats;
  words: _words;
begin
  saveSettings;

  assignFile(f2, 'files/normalWords.dat');
  rewrite(f2);
  for i := 0 to length(self.normalWords.list) - 1 do
  Begin
    words.word := normalWords.list[i].word;
    words.weight := normalWords.list[i].weight;
    write(f2, words);
  End;
  closeFile(f2);

  assignFile(f, 'files/categoriesList.dat');
  rewrite(f);
  for i := 0 to length(categories) - 1 do
  Begin
    writeln(f, categories[i].name);
  End;
  close(f);

  for i := 0 to length(categories) - 1 do
  Begin
    assignFile(f1, 'files/' + categories[i].name + '.dat');
    rewrite(f1);
    cats.name := categories[i].name;
    cats.UDKcode := categories[i].UDKcode;
    write(f1, cats);
    closeFile(f1);

    assignFile(f2, 'files/' + categories[i].name + 'Words.dat');
    rewrite(f2);
    for j := 0 to length(categories[i].wordsList.list) - 1 do
    Begin
      words.word := categories[i].wordsList.list[j].word;
      words.weight := categories[i].wordsList.list[j].weight;
      write(f2, words);
    End;
    closeFile(f2);
  End;

end;

procedure Tcontroller.setNormalWords;
var
  path: string;
begin
  path := self.settings.normalWordsPath;
  if path <> '' then
```

```
  Begin
    self.normalWords.Free;
    self.normalWords := Twords.create;
    self.normalWords.openFile(path);
  End;
end;

procedure Tcontroller.loadSettings;
var
  f: file of _settings;
begin
  if FileExists('files/settings.dat') then
  Begin
    assignFile(f, 'files/settings.dat');
    reset(f);
    while not eof(f) do
      read(f, self.settings);
    closeFile(f);
  End;
end;

procedure Tcontroller.saveSettings;
var
  f: file of _settings;
begin
  statusLabelGlobal.Caption := 'Сохранение настроек...';
  assignFile(f, 'files/settings.dat');
  rewrite(f);
  write(f, settings);
  closeFile(f);
  statusLabelGlobal.Caption := '';
end;

function Tcontroller.CompareWords(word1: string; word2: string;
  minpercent: real): real;
var
  len, lenf, minint: integer;
  Exit: boolean;
begin
  len := length(word1);
  if length(word1) > length(word2) then
  Begin
    // writeln(word1,'>',word2);
    word1 := copy(word1, 0, length(word2));
    len := length(word2);
  End
  else
  Begin
    // writeln(word1,'<',word2);
    word2 := copy(word2, 0, length(word1));
    len := length(word1);
  End;
  lenf := len;
  minint := round((len * minpercent) / 100);
  // writeln('len: ', len, ' minint: ', minint, ' ', word1, ' ', word2);
```

```
  Exit := false;
  repeat
    if AnsiCompareText(copy(word1, 0, len), copy(word2, 0, len)) = 0 then
      Exit := True;
    if Exit = false then
      len := len - 1;
    if len < minint then
      Exit := True;
  until Exit;
  // writeln(len);
  if (len >= minint) and (lenf > 0) then
    result := (100 * len) / lenf
  else
    result := -1;
end;

procedure Tcontroller.removeNormalWords(index: integer);
var
  i, j: integer;
begin
  for i := 0 to length(normalWords.list) - 1 do
    for j := 0 to length(categories[index].wordsList.list) - 1 do
      if CompareWords(normalWords.list[i].word,
        categories[index].wordsList.list[j].word, settings.comparePercent)
        > -1 then
        categories[index].wordsList.deleteWord(j);
end;

procedure Tcontroller.ScanText(memo:Tmemo);
var
  percent: array of real;
  scanWords: Twords;
  i, j, k, maxindex: integer;
  maxpercent:real;
  startTime,stopTime,path:string;
begin
  scanWords := Twords.create;
  startTime := TimeToStr(Now)+' '+DateToStr(Now);
  path := self.openDialog;
  if path<>'' then
Begin
  scanWords.openFile(path);
  for j := 0 to length(self.categories) - 1 do
  Begin
    setlength(percent, length(percent) + 1);
    if length(self.categories[j].wordsList.list) > 0 then
      for i := 0 to length(scanWords.list) - 1 do
        for k := 0 to length(self.categories[j].wordsList.list) - 1 do
          if self.CompareWords(scanWords.list[i].word,
            self.categories[j].wordsList.list[k].word,
            self.settings.comparePercent) > -1 then
            percent[length(percent) - 1] := percent[length(percent) - 1] + 1;
  End;
```

```
          for i := 0 to length(percent) - 1 do
            if length(self.categories[i].wordsList.list)>0 then percent[i]:= percent[i] /
length(self.categories[i].wordsList.list);

          maxpercent := -1;
          maxindex := -1;
          for i := 0 to length(percent) - 1 do
            if percent[i] > maxpercent then
            Begin
              maxpercent := percent[i];
              maxindex := i;
            End;
          stopTime := TimeToStr(Now)+' '+DateToStr(Now);
          memo.Lines.Clear();
          memo.Lines.Add('Реузльтаты сканирования');
          memo.Lines.Add('источник текста: '+path);
          memo.Lines.Add('Категория текста: '+ self.categories[maxindex].name);
          memo.Lines.Add('УДК код категории: '+ inttostr(self.categories[maxindex].UDKcode));
          memo.Lines.Add('Определение категори начато в: '+startTime);
          memo.Lines.Add('Определение категори закончено в: '+stopTime);

          form1.SysInfo.Lines.Clear;
          form1.SysInfo.Lines.Add('Подробная информация:');
          form1.SysInfo.Lines.Add('');
          form1.SysInfo.Lines.Add('Число схожести книги с категориями');
          for i := 0 to length(percent) - 1 do
           if percent[i] <> 0 then form1.SysInfo.Lines.Add(self.categories[i].name + ': ' +
FloatToStrF(percent[i], ffGeneral, 4, 2));
          form1.SysInfo.Lines.Add('');
          form1.SysInfo.Lines.Add('Список слов классифицируемой книги:');
          for i := 0 to length(scanWords.list) - 1 do
           form1.SysInfo.Lines.Add(scanWords.list[i].word);
        End;

        end;

        end.
```

Юнит TcategoryUnit.pas

```
        unit TcategoryUnit;

        interface

        uses
        Classes,TwordsUnit;

        type
        Tcategory = class
          name:string[255];
          UDKcode:integer;
          wordsList:Twords;
          constructor create(_name:string;_UDKcode:integer;_wordsList: Twords);
```

```
end;


implementation

 constructor Tcategory.create(_name:string;_UDKcode:integer;_wordsList: Twords);
 begin
  name:=_name;
  UDKcode:=_UDKcode;
  if _wordsList <> nil then wordsList:= _wordsList else wordsList:=Twords.create;
 end;

end.
```

Юнит TwordsUnit.pas

```
unit TwordsUnit;

interface

uses SysUtils, regexpr, Dialogs, StdCtrls, ComObj, classes;

type
  _words = record
    word: string[255];
    weight: integer;
  end;

  Twords = class
    minLengthWords: integer;
    list: array of _words;
    procedure openFile(path: string);
    procedure addWords(str: string);
    procedure deleteWord(index: integer);
    procedure RemoveDuplicates;
    procedure getWords(var list: array of string);
    procedure Clear;
  end;

implementation

uses
bookClass;

procedure Twords.addWords(str: string);
var
  i, d: integer;
  bufferstr: string;
  r: TRegExpr;
begin
  minLengthWords:=3;
  r := TRegExpr.create;
  r.Expression := '([А-Яа-яA-Za-z]{' + inttostr(minLengthWords) + ',})';
  // что искать
  if r.Exec(str) then
```

```
  begin
    repeat
      setlength(list, length(list) + 1);
      list[length(list) - 1].word := AnsiLowerCase(r.Match[1]);
    until not r.ExecNext;
  end;
  RemoveDuplicates;
end;

procedure Twords.openFile(path: string);
var
  i: integer;
  wdApp, wdDocs, wdDoc: Variant;
  str, buffer: string;
  f: textfile;
begin

  if pos('.txt', path) > 0 then
  Begin
    StatusLabelGlobal.Caption := 'открытие файла ('+path+')...';
    assignFile(f, path);
    reset(f);

    StatusLabelGlobal.Caption := '';

    i:=0;
    while not Eof(f) do
    Begin
      readln(f, buffer);
      str := str + ' ' + buffer;
    End;
    closeFile(f);
    self.addWords(str);
  End;
  if pos('.doc', path) > 0 then
  Begin
    // log.caption := 'Log: открытие файла Word...';
    StatusLabelGlobal.Caption := 'открытие word...';
    try
      wdApp := CreateOleObject('Word.Application');
    except
      ShowMessage('Не удалось запустить MS Word. Действие отменено.');
      Exit;
    end;
    // log.caption:='Log: извлечение текста...';
    // Если требуется, делаем видимым окно MS Word.
    wdApp.Visible := false;
    // Ссылка на коллекцию документов.
    wdDocs := wdApp.Documents;
    // Попытка открыть выбранный файл и получить ссылку на объект документа.
    StatusLabelGlobal.Caption := 'открытие файла ('+path+')...';
    wdDoc := wdDocs.Open(path);
    // Читаем текст документа и записываем его в Мемо.
    str := wdDoc.Range.Text;
    // Закрываем MS Word.
```

```
    wdApp.Quit;
    // log.caption:='Log:';
    // Cls();
    StatusLabelGlobal.Caption := '';
    self.addWords(str);
  End;
end;

procedure Twords.RemoveDuplicates;
var
  buffer: Tstringlist;
  cnt, i: integer;
begin
  buffer := Tstringlist.create;
  try
    buffer.Sorted := True;
    buffer.Duplicates := dupIgnore;
    buffer.BeginUpdate;
    for cnt := 0 to length(list) - 1 do
      buffer.Add(list[cnt].word);
    buffer.EndUpdate;
    setlength(list, 0);
    for i := 0 to buffer.count - 1 do
    Begin
      setlength(list, length(list) + 1);
      list[length(list) - 1].word := buffer[i];
    End;
  finally
    FreeandNil(buffer);
  end;
end;

procedure Twords.deleteWord(index: integer);
var
  i: integer;
begin
  for i := index to length(list) - 2 do
    list[i] := list[i + 1];
  setlength(list, length(list) - 1);
end;

procedure Twords.getWords(var list: array of string);
var
  i: integer;
begin

end;

procedure Twords.Clear;
begin
  setlength(self.list,0);
end;

end.
```